RENDERING THROUGH ITERATED FUNCTION SYSTEMS

HUW JONES AND MAGNUS MOAR
Centre for Electronic Arts, Middlesex University, Cat Hill, Barnet EN4 8HT, United Kingdom
E-mail: <d.h.jones, m.moar>@mdx.ac.uk

Iterated Function Systems (IFS) have been widely used for image compression and for generating fractal objects. Using barycentric coordinates and an extension of the concept, a range of rendering methods, such as Lambert, Gouraud and Phong shading, can be generated using IFS. By rendering a scattered collection of individual points using z-buffer and shadow buffer, the problems of clipping, hidden surface elimination and shadow generation are reduced to very simple forms. The method is relatively simple to program, but will not achieve the high speeds of current sophisticated rendering methods.

1 Iterated functions systems and the chaos game

Iterated function systems (IFS) have been used to generate models of fractal objects[1, 2] and for image compression[3, 4]. From the definitions of relatively few functions, often complex and unique figures can be drawn. Given n contractive transformations $\{f_0, f_1, f_2, \ldots f_{n-1}\}$ and an arbitrary starting point, a unique image in 2D or object in 3D is created by:

Given an initial point P_0
Loop for i = 1 to a large value
  Select function f_i at random from $\{f_0, f_1, \ldots f_{n-1}\}$
  Set point $P_i = f_i(P_{i-1})$
  Plot point P_i
End loop

This algorithm, often referred to as 'the chaos game', generates a Markov chain[5] of points, each new point dependent only on the previous one. Some stray points may be created initially (avoided by looping a few times before plotting or by setting P_0 inside the object), but points are soon 'attracted' into the image or object defined, the 'attractor' cannot then be escaped. Contractive transformations reduce the distance between distinct points. Uniform point density for each transformation is achieved when the probability of selection is proportional to contraction ratios (the fractional area or volume reduction), but uniform point density within the image is only achieved if the transformations have non-overlapping image sets.

The theoretical development of IFS[1, 2] and most subsequent uses concentrate on *affine* transformations, familiar to computer graphics experts as combinations of translations, scalings and rotations[6]. These standard rules have been relaxed in a number of studies, for example, Gröller[7] added 'tapering' and 'twisting' functions, Frame and Angers[8] used higher level polynomials, Jones and Campa[9] used randomised functions and Jones and Moar[10] used functions involving moduli. It has been shown by Hart[11] that, 'even the contractivity condition can be weakened to so called eventual contractivity'.

A particularly simple example of the 'chaos game' that generates a Sierpinski gasket or triangle (Fig. 1) is

Given three vertices V_0, V_1, V_2
Set point $P_0 = V_0$
Loop for i = 1 to a large value
  Select vertex V_i at random from $\{V_0, V_1, V_2\}$
  Set point $P_i = 0.5(V_i + P_{i-1})$
  Plot point P_i
End loop

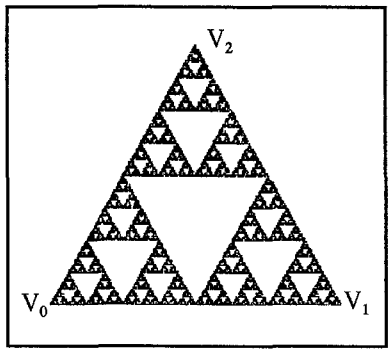

Figure 1. A Sierpinski gasket, based on an equilateral triangle, created by the chaos game algorithm. The method produces similar results for any form of triangle.

Point P_i is half way between the previous point P_{i-1} and the randomly selected vertex V_i. Setting $P_0 = V_0$ puts the initial point inside the attractor, so no stray points are created. As well as regular geometric forms, IFS can successfully create plant like images exhibiting self similarity[2, 10].

2 Barycentric coordinates

A point P in a triangle $V_0V_1V_2$ (Fig. 2) can be represented by barycentric coordinates (a, b) where

$$P = V_0 + a(V_1 - V_0) + b(V_2 - V_0),$$

using P, V_0, and so on as position vectors. This is a unique definition of P, with $a \geq 0$, $b \geq 0$ and $(a + b) \leq 1$.

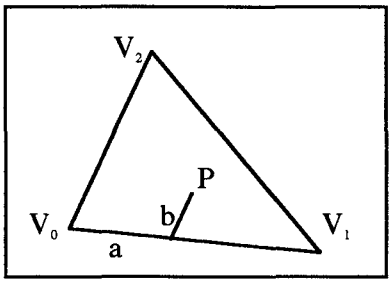

Figure 2. Barycentric coordinates (a, b) identify an interior point in a triangle.

These conditions are easily seen by identifying barycentric coordinates as a mapping of the general triangle $V_0V_1V_2$ to $V'_0V'_1V'_2$, where $V'_0 = (0, 0)$, $V'_1 = (1, 0)$ and $V'_2 = (0, 1)$, in a 2D Cartesian space (Fig. 3).

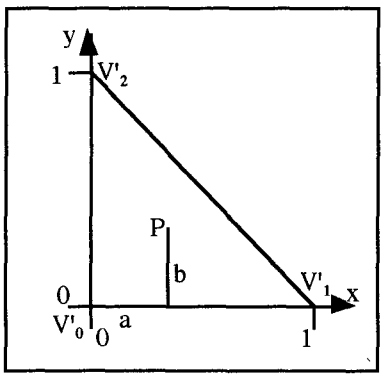

Figure 3. Mapping of the triangle $V_0V_1V_2$ into a Cartesian space to show conditions on a and b.

a and b are the fractions of edges V_0V_1 and V_0V_2 that are traversed in order to reach P from V_0. Rearranging the definition of P gives its position in terms of the triangle vertices as

$$P = (1 - a - b)V_0 + aV_1 + bV_2,$$

clearly a weighted mean of V_0, V_1, V_2 with weights $(1 - a - b)$, a and b. Conditions on a and b show these weights are non-negative and add to one, so P is the centre of mass of $(1 - a - b)$, a and b placed at V_0, V_1, V_2 respectively. This is analogous to use of blending functions in spline curves, is independent of order of selection of vertices and has implications for triangle shading.

3 The chaos game and barycentric coordinates

The functions implicit in the 'chaos game' map points from within the initial triangle to a half scaled triangle located at the randomly selected vertex, for example when vertex V_0 is selected, Fig. 4 shows the shaded half scale triangle, $V_0W_2W_1$ (W_i is the edge mid point opposite V_i). In the chaos game, if P_{i-1} has barycentric coordinates (a, b),

$$P_{i-1} = (1 - a - b)V_0 + aV_1 + bV_2,$$

the barycentric coordinates of the subsequent point $P_i = 0.5(V_i + P_{i-1})$ can be found for each selection of V_i.

1. When $V_i = V_0$,
   $P_i = (1 - 0.5a - 0.5b)V_0 + 0.5aV_1 + 0.5bV_2$,
   so the barycentric coordinates are $(0.5a, 0.5b)$;
2. when $V_i = V_1$,
   $P_i = (0.5 - 0.5a - 0.5b)V_0 + 0.5(1 + a)V_1 + 0.5bV_2$
   $= (1 - 0.5(1 + a) - 0.5b)V_0 + 0.5(1 + a)V_1 + 0.5bV_2$,
   so the barycentric coordinates are $(0.5(1 + a), 0.5b)$;

3. when $V_i = V_2$,
   $P_i = (0.5 - 0.5a - 0.5b)V_0 + 0.5aV_1 + 0.5(1 + b)V_2$,
   $= (1 - 0.5a - 0.5(1 + b))V_0 + 0.5aV_1 + 0.5(1 + b)V_2$,
   so the barycentric coordinates are $(0.5a, 0.5(1 + b))$.

Thus, the version of the chaos game that generates a Sierpinski triangle can be rewritten as:

```
Given three vertices V0, V1, V2
Set a = 0, b = 0
Loop for i = 1 to a large value
  Select j at random from {0, 1, 2}
  Case
    j = 0: set a = 0.5a, b = 0.5b;
    j = 1: set a = 0.5(1 + a), b = 0.5b;
    j = 2: set a = 0.5a, b = 0.5(1 + b);
  End case
  Set point Pi to have barycentric coordinates (a, b)
  Plot point Pi
End loop
```

This is slightly more complicated than the original form, but has interesting implications discussed later. Note that multiplication by 0.5 may be implemented very efficiently in most programming languages.

4 Rendering by IFS

4.1 Triangle rendering

Standard rendering algorithms 'fill' a polygon with colour. Some algorithms require that all the polygon faces are reduced to sets of triangles, so the triangle is the essential shape. The chaos game applied to triangle $V_0V_1V_2$ maps the original triangle to each of the sub-triangles $V_0W_2W_1$, $W_2V_1W_0$ and $W_1W_0V_2$ leaving internal triangle $W_0W_1W_2$ unfilled (Figs. 1, 5). The 'collage theorem'[2] indicates that a shape is generated by covering it with copies of itself, so to fill $V_0V_1V_2$, a fourth transformation is needed to map itself into the half scale congruent triangle $W_0W_1W_2$. The point P with barycentric coordinates (a, b) in $V_0V_1V_2$ should be mapped into P' with the same barycentric coordinates (a, b) in $W_0W_1W_2$ (Fig. 5). This sets

$$P' = (1 - a - b)W_0 + aW_1 + bW_2.$$

The W_i are the mid points of triangle edges, so

$$P' = 0.5(1 - a - b)(V_1 + V_2) + 0.5a(V_0 + V_2) + 0.5b(V_0 + V_1),$$

which can be rearranged as

$$P' = (1 - 0.5(1 - a) - 0.5(1 - b))V_0 + 0.5(1 - a)V_1 + 0.5(1 - b)V_2,$$

showing that the barycentric coordinates of P' in the original triangle $V_0V_1V_2$ are $(0.5(1 - a), 0.5(1 - b))$. This again finds the point P' as a weighted mean of triangle vertices V_0, V_1, V_2 with weights $0.5(a + b)$, $0.5(1 - a)$ and $0.5(1 - b)$ respectively. The weights are

all between 0 and 1 due to similar conditions on a and b, and they add to one, so this is a valid weighted mean. With equal probabilities of selection of non-overlapping transformations, each with equal contraction ratio of 0.5, triangle $V_0V_1V_2$ is filled with uniform point density by the following adapted chaos game.

```
Given three vertices V0, V1, V2
Set a = 0, b = 0                    // arbitrarily chooses V0 as the first point
Loop for i = 1 to a large value
  Select j at random from {0, 1, 2, 3}
  Case
    j = 0: set a = 0.5a, b = 0.5b;
    j = 1: set a = 0.5(1 + a), b = 0.5b;
    j = 2: set a = 0.5a, b = 0.5(1 + b);
    j = 3: set a = 0.5(1 – a), b = 0.5(1 – b);
  End case
  Set point Pi to have barycentric coordinates (a, b)
  Plot point Pi
End loop
```

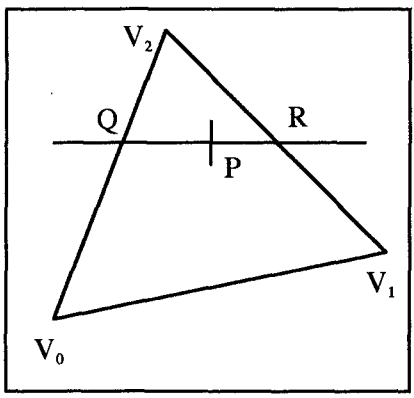

Figure 4. Scan line rendering: the internal point P lies on a scan line through Q and R.

In scan line rendering, the colours of interior points of polygons are calculated by linear interpolation, initially along edges and then along a scan line, of pre-calculated colour (for Gouraud shading) or surface normals (for Phong shading) at polygon vertices[6]. In Fig. 4, a scan line cuts the 'active' edges V_2V_0 and V_2V_1 at Q and R. If $I(V_i)$ is the colour intensity at Vi, the colour at Q is

$$I(Q) = (1 - q)I(V_2) + qI(V_0),$$

where $q = V_2Q/V_2V_0$. With $r = V_2R/V_2V_1$

$$I(R) = (1 - r)I(V_2) + rI(V_1),$$

and with $p = QP/QR$,

$$I(P) = (1 - p)I(Q) + pI(R).$$

In practice, these formulae are used to pre-calculate offset colour changes and positional changes of P, Q and R for horizontal and vertical pixel moves.

The method has a slight distorting effect, as the perspective projection of the 3D triangle is non-affine. Points closer to the observer should display smaller colour change per pixel, but the method gives a uniform change in the image space, rather than in the world space[6]. The method is, in any case, a distortion of the correct physical model. The algorithms for Gouraud and Phong shading allow updating of the active edge list when the scan line passes a vertex, so 'spans' like QR can be easily identified and updated for polygons other than triangles.

For a triangle with an interior point P known through its barycentric coordinates (a, b), colour can be established directly from the weighted average of the colour (or surface normal) values at the vertices, without reference to scan line directions. For Gouraud shading,

$$I(P) = (1 - a - b)I(V_0) + aI(V_1) + bI(V_2).$$

For Phong shading, the mean surface normal vector is

$$\mathbf{n}(P) = \text{Normalise}[(1 - a - b)\mathbf{n}(V_0) + a\mathbf{n}(V_1) + b\mathbf{n}(V_2)],$$

where $\mathbf{n}(P)$ is the surface normal allocated to point P, and Normalise is a function to reduce a vector to unit length. The argument of a 'Normalise' function cannot have zero length, this would only occur in the case of a degenerate or non-manifold object.

This gives an easy way to fill triangles using IFS with barycentric coordinates, which can be shown to give exactly the same results as standard Gouraud and Phong shading. The barycentric method is independent of the scan line direction, and this equivalence indicates similar independence of the traditional methods from the scan line direction for *triangle* shading, which is not the case for polygons with more edges.

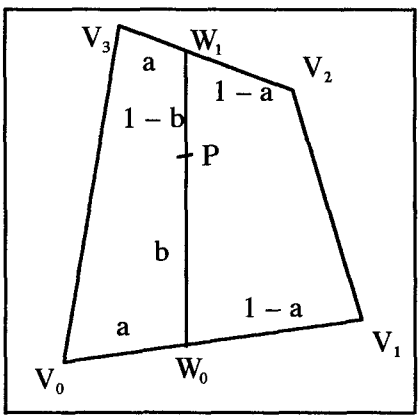

Figure 5. Extended barycentric coordinates identify an inerior point P in a quadrilateral.

4.2 Quadrilateral rendering

We can identify an internal point P of a quadrilateral $V_0V_1V_2V_3$ using 'extended barycentric coordinates' (a, b). There is a unique line W_0W_1 which passes through P so that the edges V_0V_1 and V_3V_2 are divided in equal proportion, a:(1 – a) (Fig. 5). In other words,

$$W_0 = (1 - a)V_0 + aV_1,$$

$$W_1 = (1 - a)V_3 + aV_2,$$

where $0 \leq a \leq 1$. If P lies a proportion b along W_0W_1

$$P = (1 - b)W_0 + bW_1,$$

where $0 \leq b \leq 1$. This effectively maps any interior point of the quadrilateral into a unit square (Fig. 6), where

$V'_0 = (0, 0)$, $V'_1 = (1, 0)$, $V'_2 = (1, 1)$ and $V'_3 = (0, 1)$.

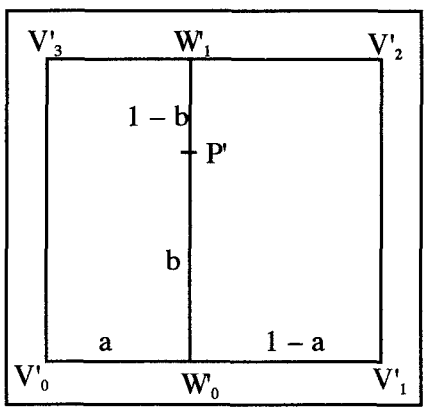

Figure 6. Mapping of the quadrilateral of figure 5 into a unit square

This is equivalent to the effect of Fig. 3 for a triangle. The meaning of an internal point is obvious for a planar convex quadrilateral (Fig. 5). For non-planar vertices, the point P lies on a bi-linear patch, generating a form of hyperbolic paraboloid[12]. For planar concave quadrilaterals, P may lie outside the triangle boundary, but within the convex hull of the vertices. Such quadrilaterals can be identified and split into triangles when rendering, although this allows the ambiguity of two possible interpretations.

Substituting for W_0 and W_1 into the expression for p gives

$$P = (1 - b)\{(1 - a)V_0 + aV_1\} + b\{(1 - a)V_3 + aV_2\},$$

$$P = (1 - a)(1 - b)V_0 + a(1 - b)V_1 + abV_2 + (1 - a)bV_3,$$

giving P as a weighted mean of the four vertices. This is valid as each of the four weights$(1 - a)(1 - b)$, $a(1 - b)$, $(1 - a)b$ and ab lies between zero and 1 from conditions on a and b, and it is easy to establish that their sum is one. This gives a direct method for proportional shading from vertex colours or normals, which is the same as standard scan line methods only when W_0W_1 happens to be aligned with a scan line (Fig. 7). The IFS method is an improvement on standard methods, in that the shading allocated to a point is not dependent on quadrilateral orientation.

The chaos game applied to the four vertices of a square (Fig. 6) will fill the square[13]. By selecting V'_i to be one of the vertices V'_0, V'_1, V'_2 or V'_3 at random, and finding the mid point of P' and V'_i, we have the essential step of the chaos game that will fill the square $V'_0V'_1V'_2V'_3$. It should be clear that the required mid points have coordinates

- P' and V'_0: (0.5a, 0.5b);
- P' and V'_1: (0.5(1 + a), 0.5b);
- P' and V'_2: (0.5(1 + a), 0.5(1 + b));
- P' and V'_3: (0.5a, 0.5(1 + b)).

Applying the usual chaos game directly to the vertices of a general quadrilateral $V_0V_1V_2V_3$ (Fig. 5) will only fill it if it is a parallelogram. When the quadrilateral is skew, a fractal 'Sierpinski tetrahedron' will result[9]. If the chaos game is applied to the extended barycentric coordinates of a general quadrilateral according to the algorithm specified below, it fills all 'interior' points as explained earlier in this section, not now with uniform density, but still giving weights that lead to easy rendering from vertex colours or normals.

```
Set (a, b) to (0, 0)
Loop for i = 1 to a large value
 Select j at random from {0, 1, 2, 3}
 Case
  j = 0: set a = 0.5a, b = 0.5b;
  j = 1: set a = 0.5(1 + a), b = 0.5b;
  j = 2: set a = 0.5(1 + a), b = 0.5(1 + b);
  j = 3: set a = 0.5a, b = 0.5(1 + b);
 End case
 Set Pi to have extended barycentric coordinates (a, b)
 Plot point Pi
End loop
```

For Gouraud type shading, we have

$$I(P) = (1 - a)(1 - b)I(V_0) + a(1 - b)I(V_1) + ab\,I(V_2) + (1 - a)bI(V_3),$$

where $I(V_i)$ is the colour allocated to vertex V_i, as above. For Phong type shading, normals are interpolated by

$$\mathbf{n}(P) = \text{Normalise}[(1 - a)(1 - b)\mathbf{n}(V_0) + a(1 - b)\mathbf{n}(V_1) + ab\mathbf{n}(V_2) + (1 - a)b\mathbf{n}(V_3)].$$

Shading depends only on vertex values, so is independent of scan line direction, unlike standard methods. Symmetry of expressions shows that the order of selection of vertices does not affect shading.

5 An example: Gouraud shading of a cone

We consider Gouraud shading of a cone as a simple example. If the cone is represented polyhedrally as a pyramid with n sloping triangular faces and a regular n-gon as its base (Fig. 7), there is a problem of how to allocate surface normals to the triangle vertices which meet at V. If we average normals for all triangles meeting at this point, the applied normal points directly out of the vertex along the cone's axis, giving the vertex an incorrect flattened shading. Using the triangle's own normal at V for each triangle avoids this, but this shows creasing along triangle edges near V, as there is no continuity across triangle edges. The singular point at V has different normals, hence shading values, close to V.

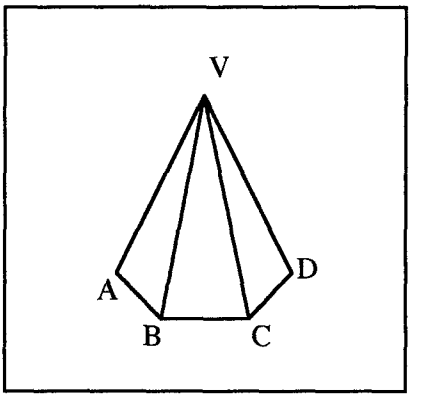

Figure 7. Polygonal approximation of a cone as a pyramid on a regular polyhedral base.

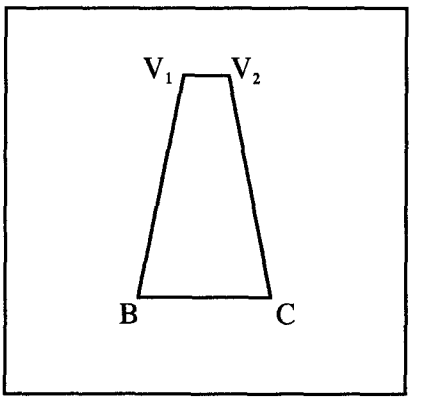

Figure 8. Representation of triangle BCV as a degenerate quadrilateral.

If vertex triangles are replaced by quadrilaterals with a zero length edge at V (Fig. 8), the problem can be overcome. At V_1, the surface normal is taken as the normalised mean of normals of faces VAB and VBC and at V_2 the normal is the normalised mean of the normals of faces VBC and VCD. If V_1 and V_2 are made coincident at V, this gives continuity at V, avoiding creasing. The now degenerate trapezium V_1BCV_2 has the same shape as the isosceles triangle VBC. There is still a singular point at V, as more than one normal is allocated to the same point, with a multiplicity of colour values possible at the vertex on rendering, but this is anti-aliased by the random nature of the IFS. Figure 9 clearly shows creasing from the standard method and its elimination by the degenerate quadrilateral method, both rendered using an IFS on an eight-sided based pyramid. (This image was rendered on a 16 bit machine, so some colour banding is seen).

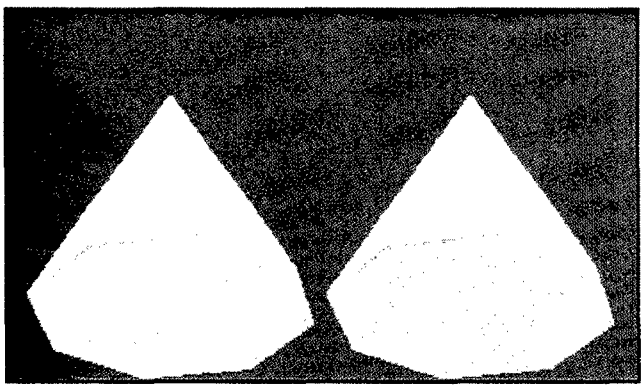

Figure 9. Gouraud shaded cones represented as octagonal based pyramids. The left image shows creasing at the vertex, the right image, which uses degenerate quadrilaterals, has a smoother representation.

6 Summary

A method for using IFS to render non-fractal forms in a way similar to standard scan line algorithms has been shown. Compared to scan line methods, IFS rendering uses different algorithms for triangles and quadrilaterals, not dealing with higher orders. The algorithm selects world points at random, so stochastic anti-aliasing occurs by painting each pixel with a sample of its possible colours. This is an advantage for still images, but may give undesirable 'boiling' on animation. Standard methods visit each pixel once per polygon, the IFS method has redundancy, with pixels being visited many times without a guarantee that all are filled. IFS methods are unlikely to challenge established methods on speed. However, programming them is relatively simple, with clipping, hidden surface and shadow problems reduced to simple decisions.

An example using Gouraud type shading on a basic shape is shown (Fig. 9). There is much scope for extension. Work on Phong type shading with specular highlights and on texture mapping, for which the method should be particularly suited, is underway at

Figure 10. IFS images showing fractal and non-fractal objects using z-buffer and shadow buffer and showing the potential of the method for texture mapping.

the time of writing. As an indication of the scope of the method, two images incorporating fractal trees (with non-affine transformations), shadows and texturing (for the trellis and mown grass effect)[10] are shown in Fig. 10.

7 Acknowledgments

The authors are grateful to their colleagues in the Centre for Electronic Arts for supplying them with a stimulating intellectual environment and for research funding from the Research Centre of the School of Art, Design and Performing Arts and of Middlesex University itself. The authors also thank the referees for their helpful and constructive comments.

References

1. Barnsley M. F., Fractal Modelling of Real World Images. In *The Science of Fractal Images*, ed. by Peitgen H. O. and Saupe D. (Springer-Verlag, New York, 1988) pp. 219–242.
2. Barnsley M. F. *Fractals Everywhere (2 ed)* (Academic Press, San Diego, 1992).
3. Fisher Y. *Fractal Encoding – Theory and Applications to Digital Images* (Springer-Verlag, New York, 1994).
4. Lu N. *Fractal Imaging* (Academic Press, San Diego, 1997).
5. Cox D. R. and Miller H. D. *The Theory of Stochastic Processes* (Methuen, London, 1965).
6. Foley J. D., van Dam A., Feiner S. K. and Hughes J. F. *Computer Graphics, Principles and Practice (2 ed)* (Addison-Wesley, Reading, 1990).
7. Gröller E. Modeling and Rendering of Nonlinear Iterated Function Systems, *Comput. & Graphics*, **18**(5), (1994) pp. 739-748.
8. Frame M. and Angers M. Some Nonlinear Iterated Function Systems. *Comput. & Graphics*, **18**(1), (1994) pp. 119-125.
9. Jones H. and Campa A. Abstract and natural forms from iterated function systems. In *Communicating with Virtual Worlds*, ed. by Magnenat Thalmann N. and Thalmann D. (Springer-Verlag, Tokyo, 1993) pp. 332-344.
10. Jones H. and Moar M. Iterated Function Systems and Non Affine Transformations: Examples in 2D and 3D, *15th Spring Conference on Computer Graphics*, (University of Bratislava, Budmerice, Slovakia, 28 April - 1 May, 1999) pp. 201-209.
11. Hart J. C. Iterated Function Systems and Recurrent Iterated Function Systems. In Fractal Models for Image Synthesis, Compression and Analysis, *ACM SIGGRAPH 1996, Course 27*, (ACM, New Orleans, 6 August 1996).
12. Cundy H. M. and Rollett A. P. *Mathematical Models (2 ed)* (Oxford University Press, London, 1961).
13. Jones H. Dürer, Gaskets and the Chaos Game, *Computer Graphics Forum*, **9**(3) (1991) pp. 327-332.